SEMESTER PROJECT INTRODUCTION

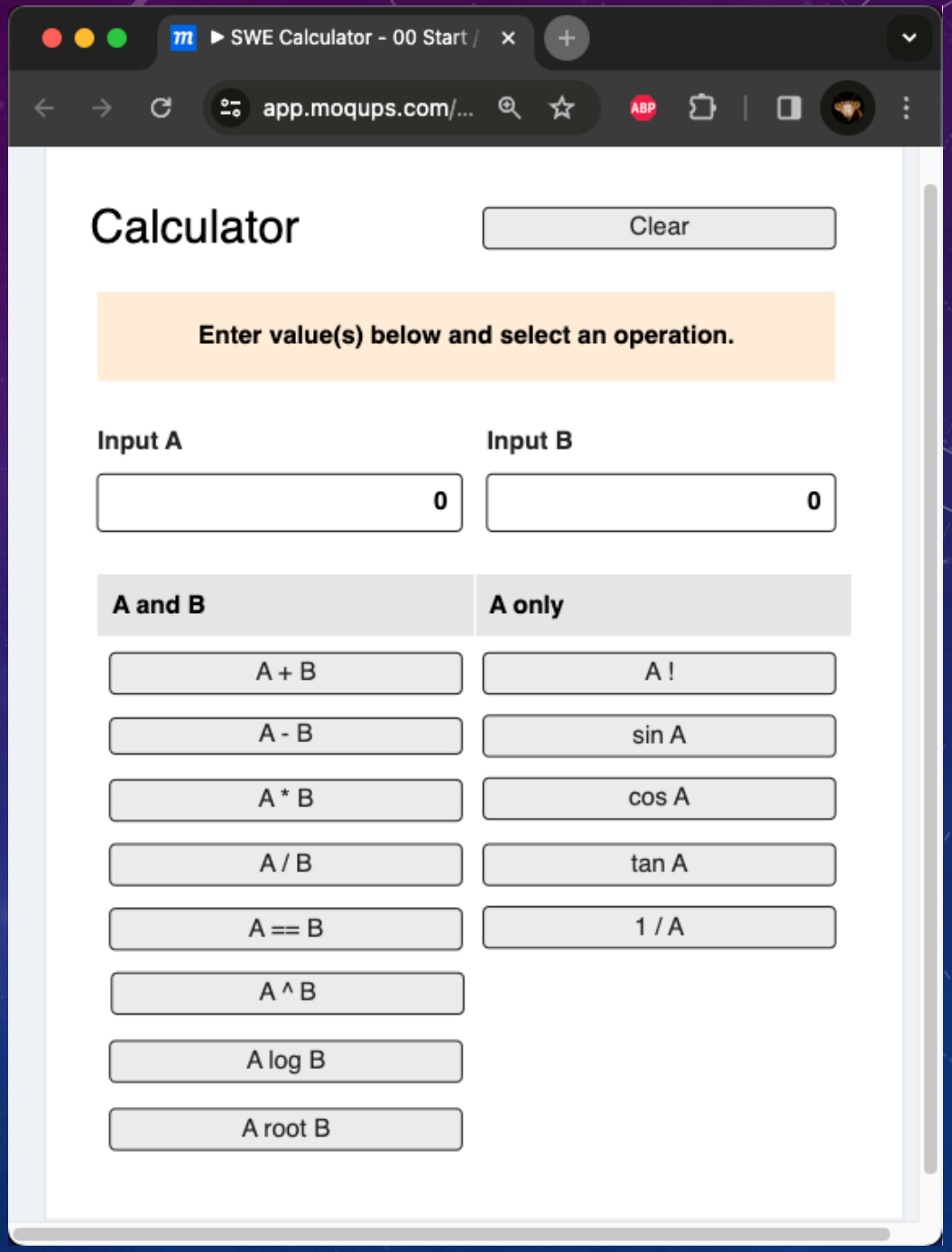
KSU SWE 3643

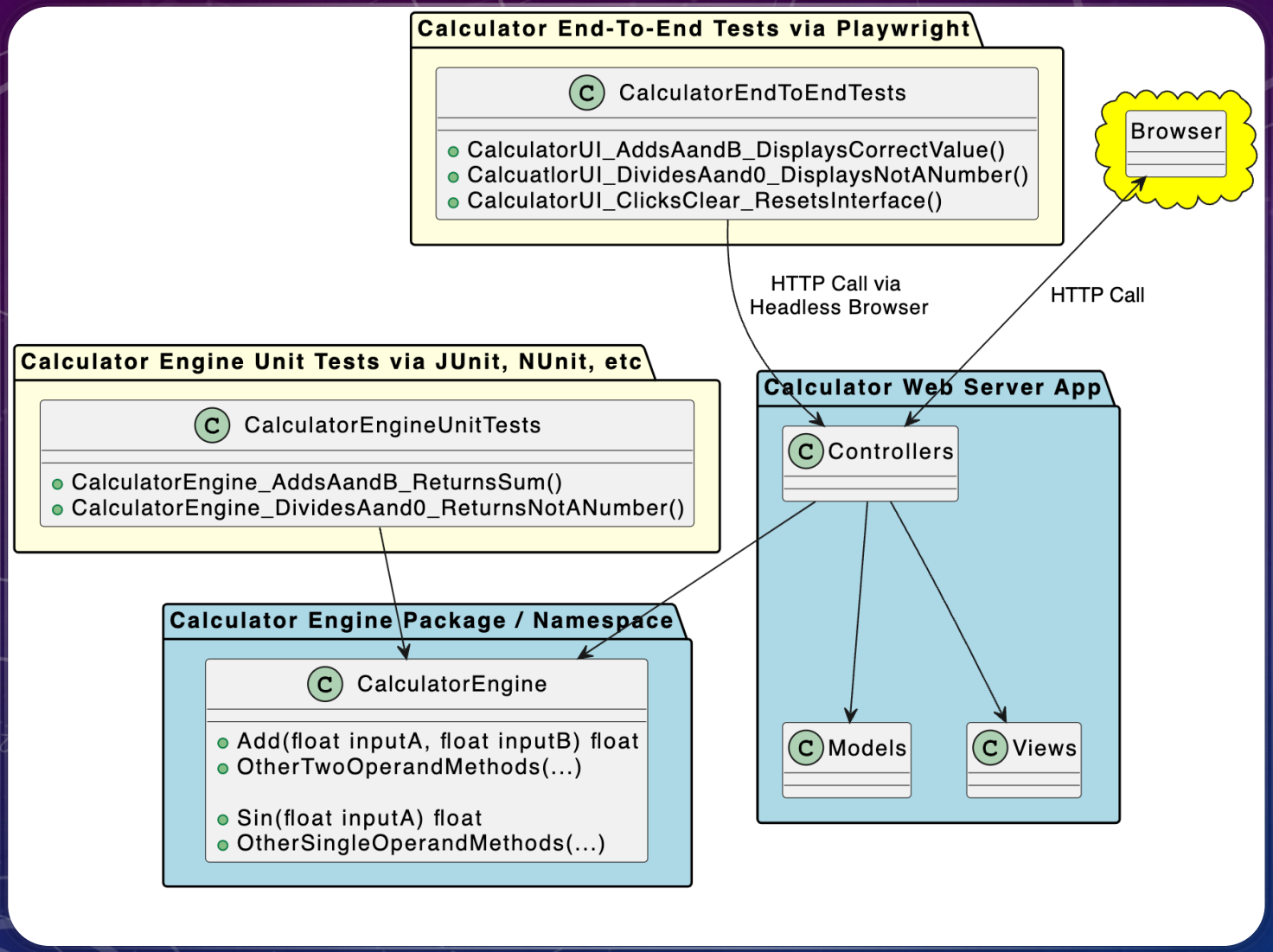
SOFTWARE TESTING AND QUALITY ASSURANCE

SPRING 2024

WEB-BASED CALCULATOR

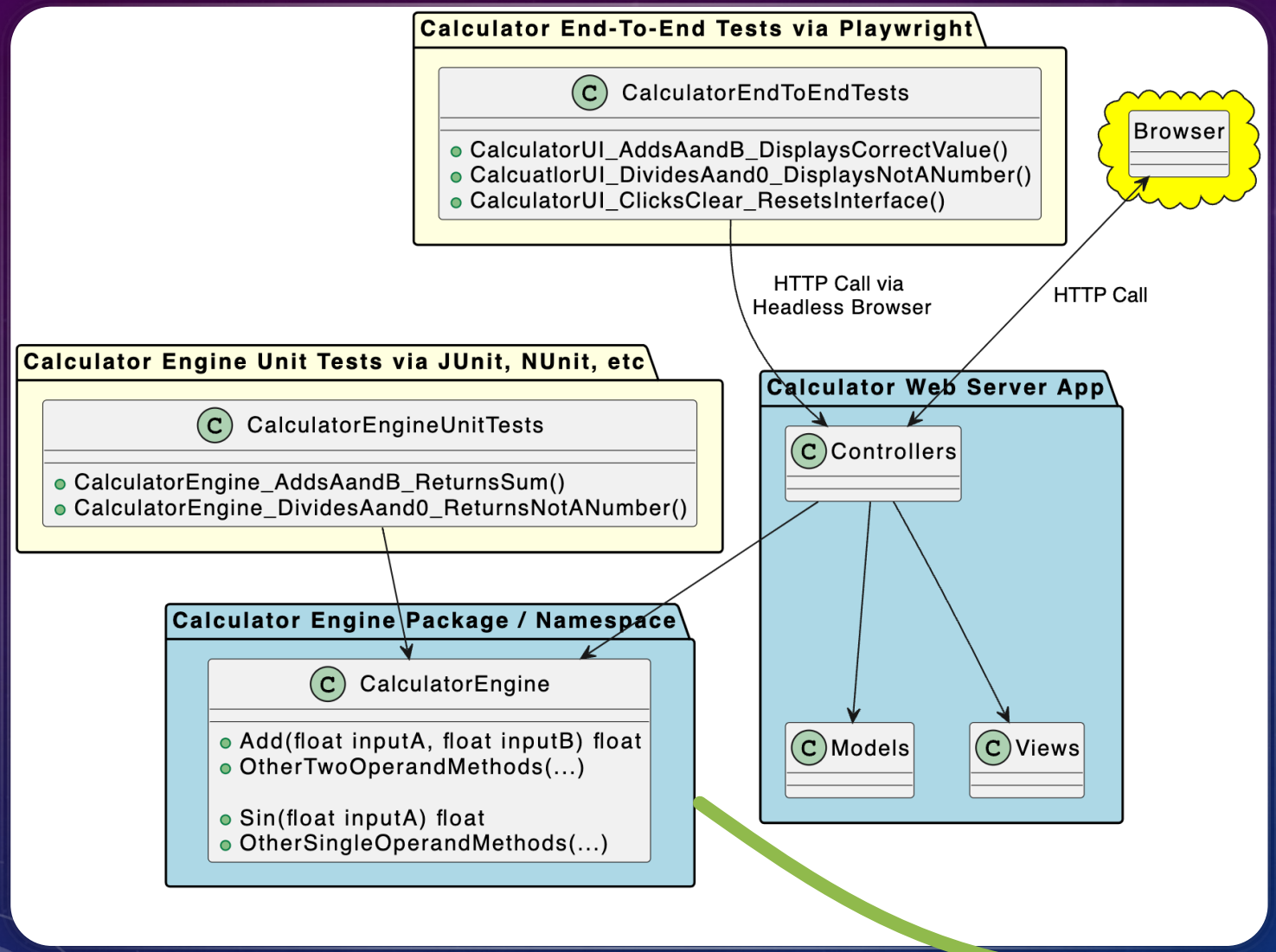
- Single and Double Operand Floating Point Calculations
- Input Validation
- Clear / Reset





ARCHITECTED FOR TESTING AND HIGH COVERAGE

ARCHITECTED FOR TESTING AND HIGH COVERAGE



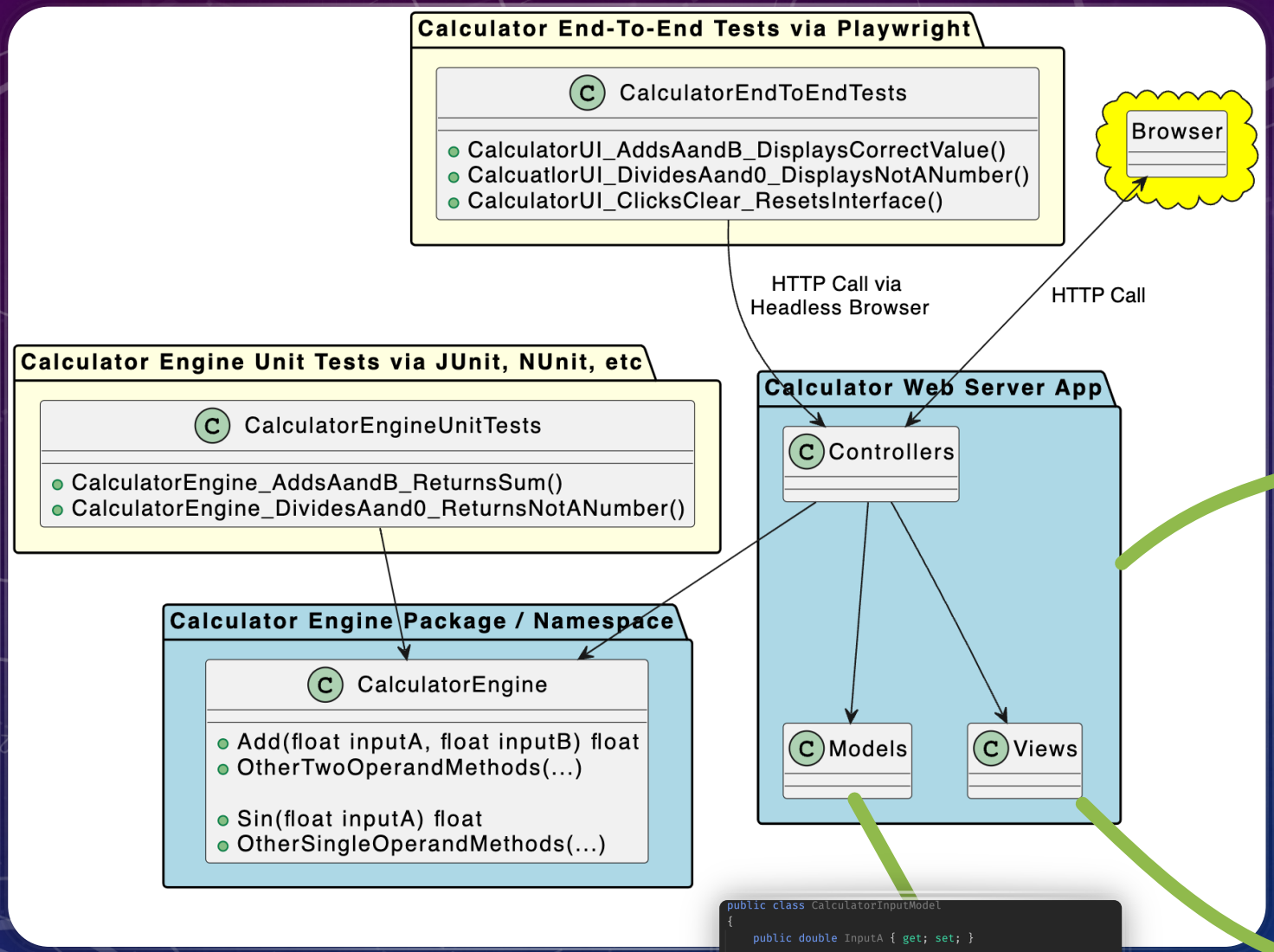
CALCULATION LOGIC IS DECOUPLED FROM PRESENTATION LOGIC

```
public class CalculationOutput(double result, bool isSuccess = true)
{
    public double Result { get; set; } = result;

    public bool IsSuccess { get; set; } = isSuccess;

    1 usage
    public List<string> ResultLines { get; set; } = [];
}

simple enough (10%)  2 usages  Jeff Adkisson *
public class BasicFloatingPointOperations {
    simple enough (10%)  1 usage  Jeff Adkisson *
    public static CalculationOutput Add(double a, double b) {
        var calculation:double = a + b;
        var resultLines = new List<string> { $"{a} + {b}", $"{calculation}" };
        return new CalculationOutput(calculation) { ResultLines = resultLines };
    }
```

Performs calculation logic, returns floating point values and errors

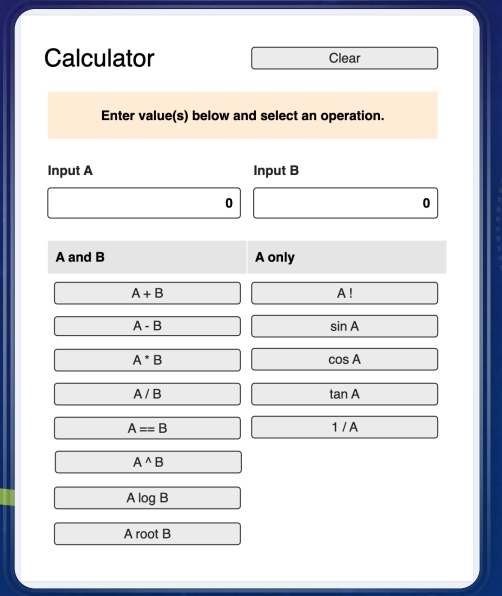
ARCHITECTED FOR TESTING AND HIGH COVERAGE

WEB SERVER APP FOLLOWS MVC PATTERN AND REFERENCES CALCULATOR LOGIC

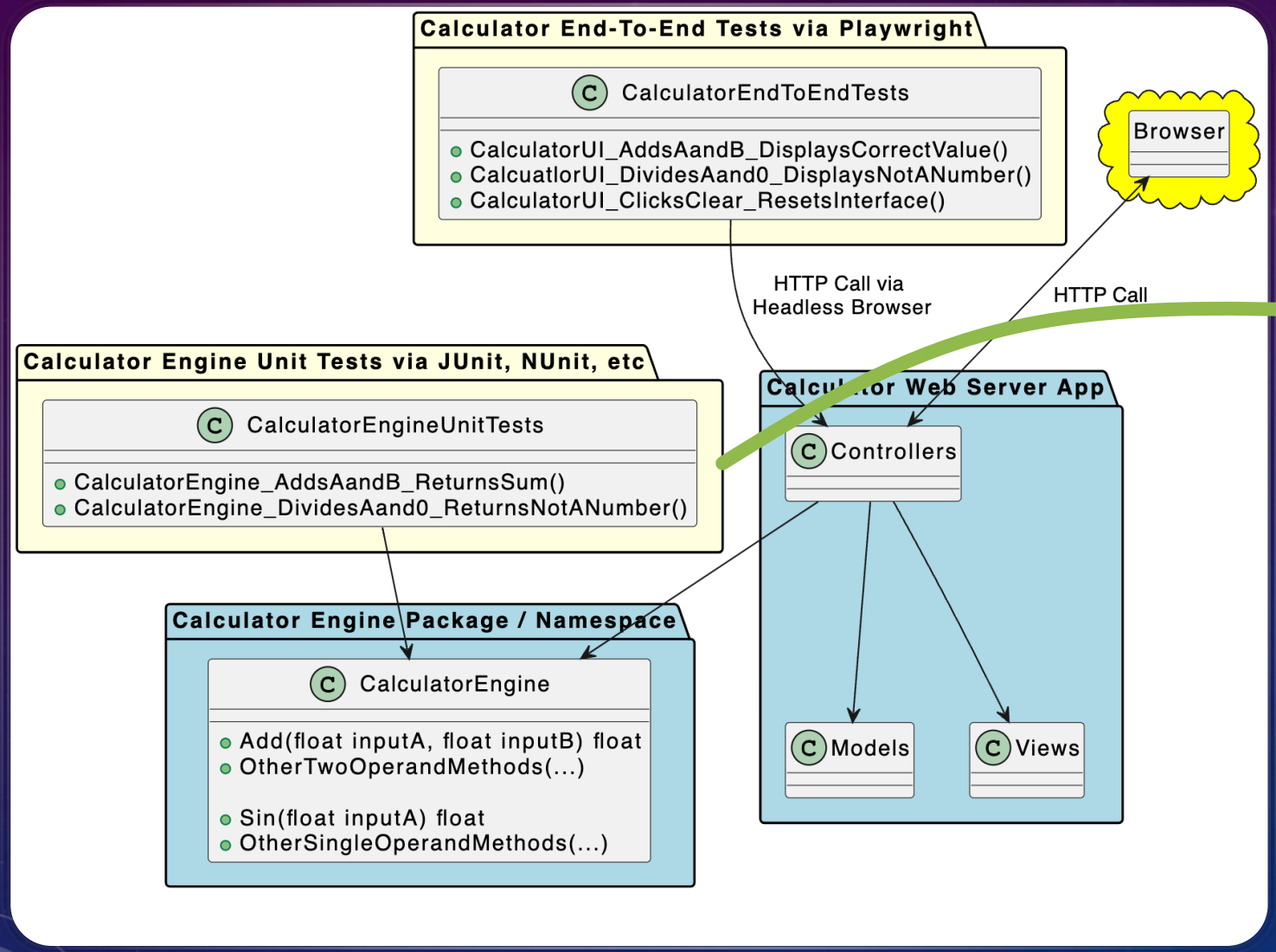


Generates the user interface, calls the Calculator Engine, and returns results to the user's browser

```
public class CalculatorInputModel
{
    public double InputA { get; set; }

    public double InputB { get; set; }

    public bool IsError { get; set; }

    public List<string> ResultLines { get; set; }
        = ["Enter value(s) below and select an operation."];
}
```



ARCHITECTED FOR TESTING AND HIGH COVERAGE

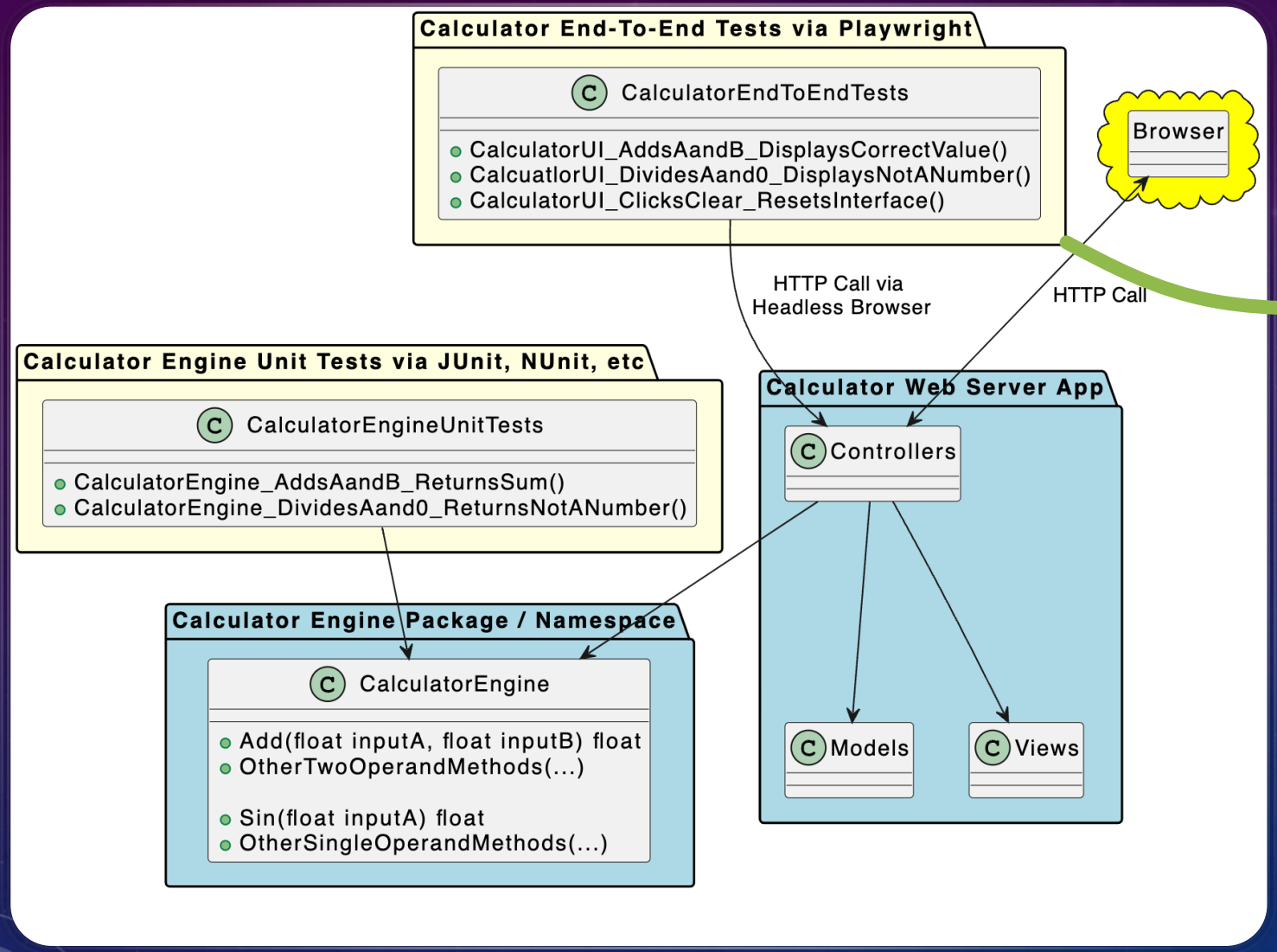


UNIT TESTS WILL ACHIEVE 100% COVERAGE OF CALCULATION LOGIC

Unit tests verify all calculation requirements meet the specification without running the application.

```
[Fact]
simple enough (10%)  ◇  Jeff Adkisson *
public void Addition_TwoFloatingPointNumbers_Adds()
{
    // Arrange
    const double a = 1.5;
    const double b = 2.75;
    const double expected = 4.26;

    // Act
    var result:double = _sut.Add(a, b);

    // Assert
    Assert.Equal(expected, actual:result, Precision);
}
```

ARCHITECTED FOR TESTING AND HIGH COVERAGE

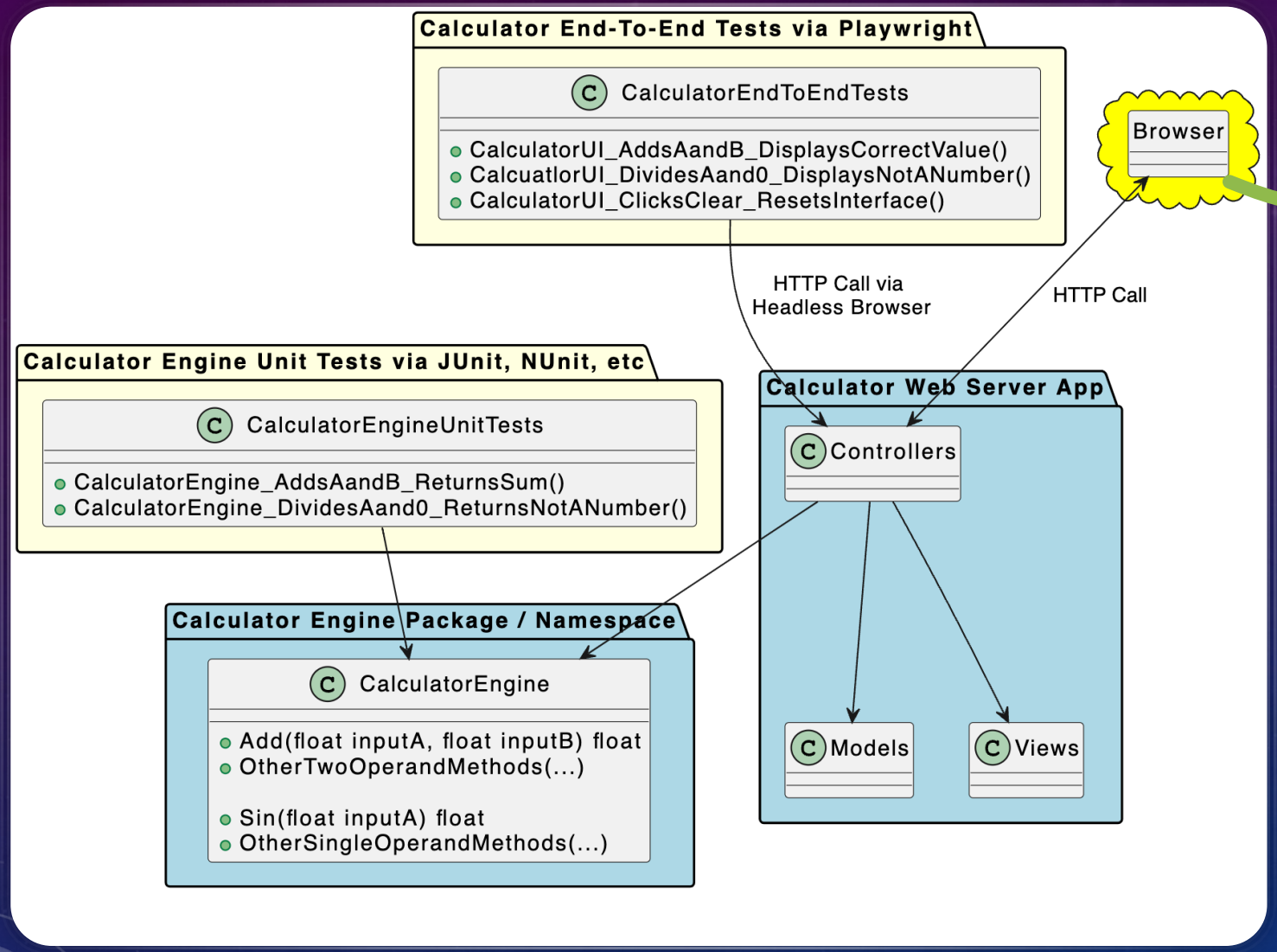


UI TESTING CAN BE VERY CHALLENGING. PLAYWRIGHT SIMPLIFIES THE PROCESS.

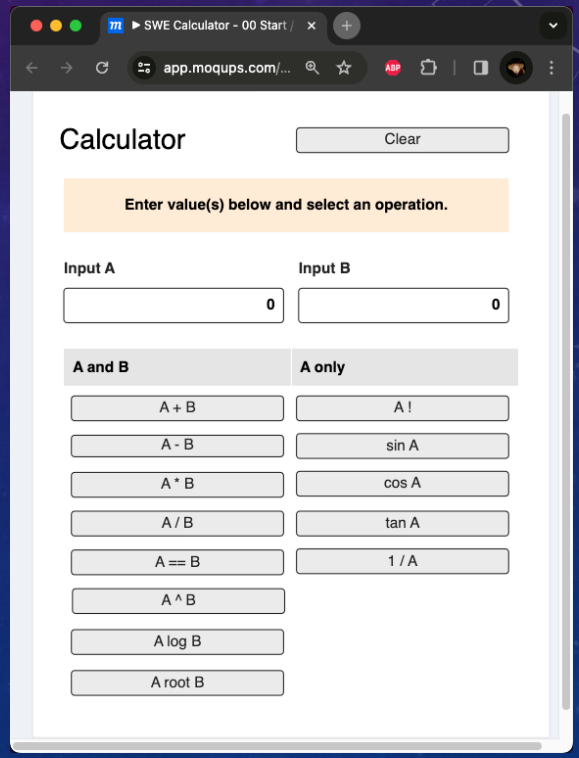
End-to-end tests via Playwright library verify UI requirements using a headless browser.

```
public class Tests : PageTest
{
    [Test]
    public async Task HomepageHasPlaywrightInTitleAndGetStartedLinkLinkingtoTheI
    {
        await Page.GotoAsync("https://playwright.dev");

        // Expect a title "to contain" a substring.
        await Expect(Page).ToHaveTitleAsync(new Regex("Playwright"));

        // create a locator
        var getStarted = Page.GetByRole(AriaRole.Link, new() { Name = "Get start

        // Expect an attribute "to be strictly equal" to the value.
        await Expect(getStarted).ToHaveAttributeAsync("href", "/docs/intro");

        // Click the get started link.
        await getStarted.ClickAsync();

        // Expects page to have a heading with the name of Installation.
        await Expect(Page
            .GetByRole(AriaRole.Heading, new() { Name = "Installation" }))
            .ToBeVisibleAsync();
    }
```

ARCHITECTED FOR TESTING AND HIGH COVERAGE



And you can run the application from a web browser.



PROJECT DELIVERABLES

- Creating a simple web-based calculator that performs basic floating point calculations on single and double operands.
- Adding unit tests to the calculator classes (the classes that perform the calculations) to achieve 100% coverage of the calculation methods.
- Performing automated end-to-end testing on your web-based UI using Playwright scripts.
- Writing documentation in Markdown describing the runtime environment, how to execute your project from the command line, and how to execute your tests from the command line.
- Producing a 5-to-8 minute video demonstrating your completed application, your unit tests and Playwright scripts from the command line.
- Checking all of your source code, tests, and documentation into GitHub.

YOUR DEVELOPMENT TEAM

Team of 1: Work Alone

You will execute the entire project alone. This is a good option if you are confident in your abilities, prefer to work alone, like to have high levels of control over your projects, or have a hard time scheduling with another person.

If you work alone, you are responsible for all the work.

This was always my preferred choice when I was a student because it left me in 100% control of my success.

Team of 2: Work with a Partner

You and a teammate will execute the entire project and share the same grade. This is a good option if you find someone who is a good complement to your abilities or shares a similar schedule as you.

Remember - you will share the same grade, so choose wisely.

Sometimes teammates let you down and you must shoulder the burden to ensure your own personal success is not impacted. The possibility exists that your teammate will drop the course. You must be ready to deliver if your teammate does not deliver.

TEAM SIGNUP BY THURSDAY, FEB 8

Team	**1**	
Team Github Repo:		
	Create a Github repository your team will share and I can reach to pull your project deliverables	
	Post the URL of the repository here. Everyone in your team must be able to commit to the repository.	
	https://docs.github.com/en/get-started/quickstart/create-a-repo	
Team Size:		*Size will be 1 or 2.*
Team Members:	**Name**	**KSU Email Address**
- Member 1		
- Member 2		
	You must use your KSU email address.	

https://shorturl.at/eAOSW

YOUR TEAM CANNOT CHANGE AFTER THURSDAY, FEBRUARY 8TH.

IF YOU HAVE NOT SIGNED UP BY FEBRUARY 8TH, YOU WILL AUTOMATICALLY BECOME A TEAM OF 1.

TEAM SIGNUP BY THURSDAY, FEB 8

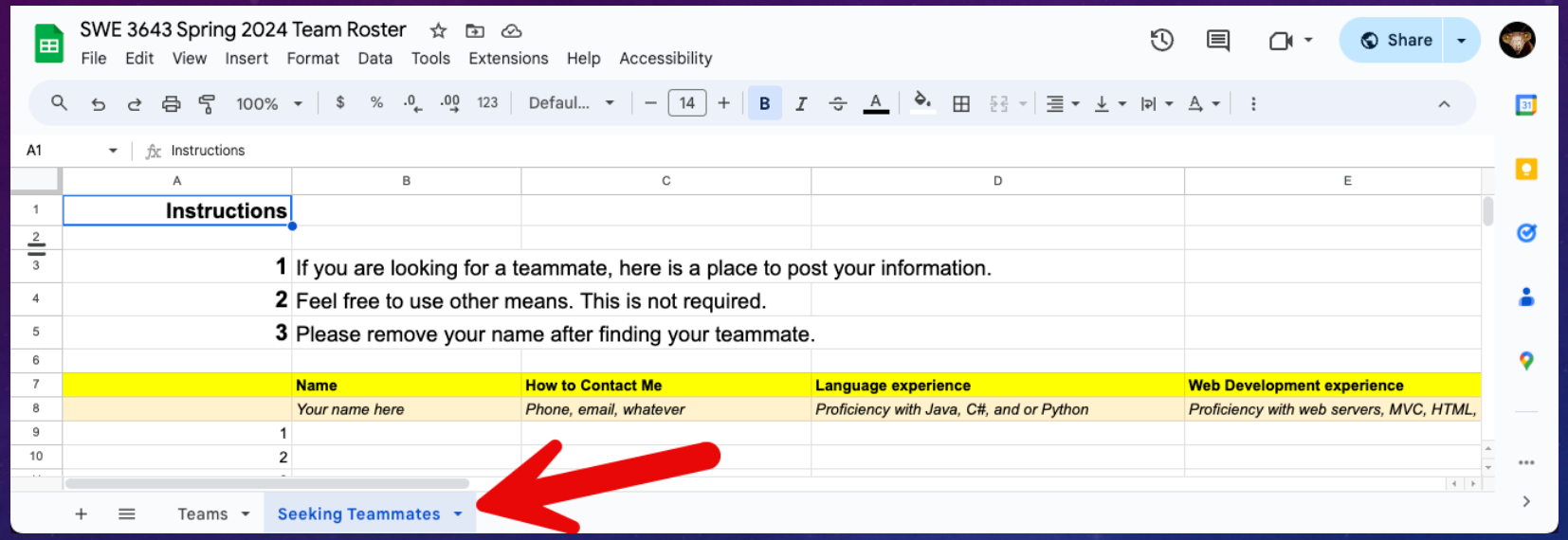
Team	**1**	
Team Github Repo:		
	Create a Github repository your team will share and I can reach to pull your project deliverables	
	Post the URL of the repository here. Everyone in your team must be able to commit to the repository.	
	https://docs.github.com/en/get-started/quickstart/create-a-repo	
Team Size:		*Size will be 1 or 2.*
Team Members:	**Name**	**KSU Email Address**
- Member 1		
- Member 2		
	You must use your KSU email address.	

https://shorturl.at/eAOSW

COMPLETING THE SIGNUP PROCESS ON TIME IS YOUR FIRST PROJECT GRADE.

IT IS WORTH 2% OF YOUR OVERALL PROJECT SCORE (EASY POINTS!)

I NEED A TEAMMATE



THE TEAM ROSTER HAS A ”SEEKING TEAMMATES” TAB YOU CAN USE.

YOU ARE WELCOME TO USE OTHER MEANS SUCH AS GROUPME, ETC.

LANGUAGES, WEB SERVER ARCHITECTURES, AND TEST RUNNERS

- **C#**
  - ASP.NET MVC + NUnit + Playwright
  - ASP.NET Blazor Server + NUnit + Playwright
- **Java**
  - Spring MVC + JUnit + Playwright
  - Struts + JUnit + Playwright
  - Grails + JUnit + Playwright
- **Python**
  - Flask + Pytest + Playwright

I KNOW REACT/ANGULAR/VUE/OTHER. CAN I USE THAT?

NO. YOU MUST USE A SERVER FRAMEWORK THAT RENDERS HTML ON THE SERVER.

SPA DEVELOPMENT IS PROHIBITED DUE TO OVERALL COMPLEXITY AND HIGHER RISK OF PROJECT FAILURE.

I LIKE PHP. CAN I USE THAT?

NO. YOUR LANGUAGE CHOICES ON THE SERVER-SIDE ARE C#, JAVA, AND PYTHON.

CAN I USE WINFORMS OR JAVAFX?

NO. END-TO-END TESTING ON WEB APPLICATIONS IS GENERALLY SIMPLER THAN END-TO-END TESTING ON DESKTOP APPLICATIONS.

I AM FREAKING OUT! I HAVE NOT DONE WEB DEVELOPMENT! WHAT SHOULD I PICK?

- Every framework listed has huge communities of developers who have posted outstanding tutorials on the web and on sites like YouTube. You will have no trouble finding resources. You must make the time to try the tutorials and learn the basics of what you choose. **This is exactly how professional engineers operate when they are assigned a new project.**
- For best results, *stick with the language you know best*. Do not pick what looks the most cool or what you think will look best on your resume… *pick what you have the best chance of performing well on the project*.
- **If you are not confident in any of the options, pick C# and Blazor Server.** I am most proficient with C# and have a lot of experience with Blazor Server, so I can provide the most help and advice with those choices.

LEARN TO BE CONFIDENT IN YOUR ABILITY TO LEARN NEW TECHNOLOGIES AND SOLVE PROBLEMS. BUDGET YOUR TIME WISELY AND DO NOT WAIT UNTIL THE LAST MINUTE TO GET STARTED. YOU CAN DO THIS.

ARE YOU GOING TO TEACH ME EVERYTHING I NEED TO KNOW?

No, but I will show you the way to success.

Your First Year instruction gave you the programming basics you need to execute this project.

You also have some experience with larger projects and working in teams from Software Engineering 1.

Finally, you have been learning problem solving approaches your entire academic career.

No one can teach you how to program but yourself. The only way to be a good programmer is to do the work.

Sometimes that will mean long, frustrating nights at your computer working out problems and endlessly researching obscure compiler errors.

Everyone in this field has had those days and nights.

I STILL FEEL NERVOUS

You must learn some things on your own, **but you are not alone**.

I am going to help you when you get stuck.

We are dedicating the last four lecture periods of the class to project work.

I will be in the classroom to help people who need advice, additional resources, etc.

EVERY SEMESTER I SEE NERVOUS STUDENTS ACHIEVE GREAT THINGS WHEN GIVEN A BIT OF A PUSH AND A LOT OF SUPPORT. **YOU CAN DO THIS.**

THIS IS A TESTING CLASS. WHY ARE WE WRITING A WEB APPLICATION?

1. You need practice building applications, learning new frameworks in short timeframes, and building the confidence that you can tackle problems like this one.
2. You need source control experience.
3. You need experience presenting your work.
4. You need to experience how to separate UI concerns from logic. Most of your student work so far has tightly coupled logic and user interface code.
5. You to need experience writing unit tests for logic you create. You will see that there are certain approaches that work better than others to effectively automate your testing.
6. You need to experience writing end-to-end tests. That is easiest to do with web frameworks.
7. One of the most pervasive problems in our field is that the development process is often separate from the test process. The Test Driven Design (TDD) process will help you write better code that is durable and easier to maintain/refactor over time.

SOURCE CONTROL VIA GIT AND GITHUB

Your project will be hosted on a public GitHub repository.

Each team will have a separate GitHub repository created by the team member(s).

git is *essential* for all software engineers regardless of industry and skill level.

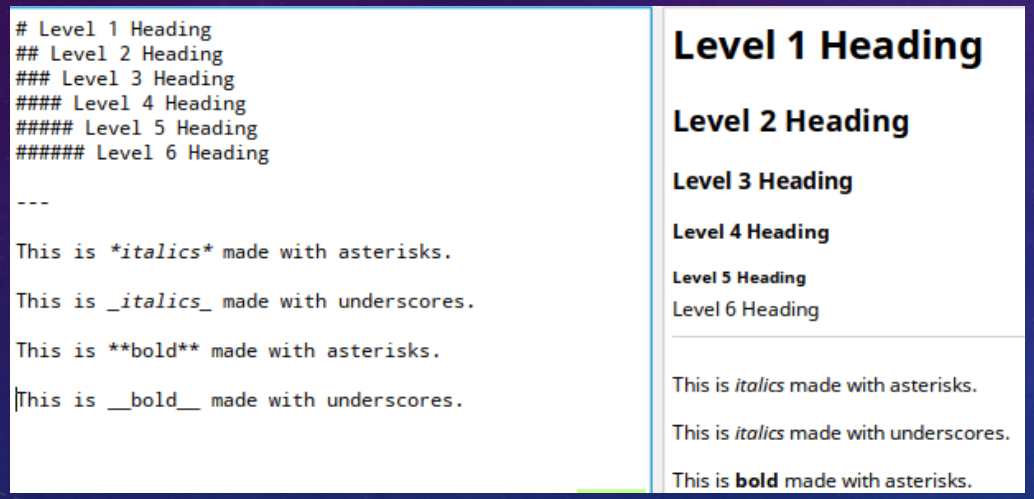
Do not graduate without expertise in source control, including proficiency in branching, merging, and pull requests.

Note that these more advanced topics are not taught by KSU, so you need to make this a personal goal. Use this project to get started learning git and carry your knowledge forward after the course ends.

THE BEST DEVELOPERS ALWAYS USE SOURCE CONTROL EVEN WHEN WORKING ALONE. IT IS AN ESSENTIAL SKILL YOU MUST DEVELOP.

DOCUMENTATION WRITTEN IN MARKDOWN

All documentation will be written in Markdown and checked into your team GitHub repository.



```
# Level 1 Heading
## Level 2 Heading
### Level 3 Heading
#### Level 4 Heading
##### Level 5 Heading
###### Level 6 Heading

---

This is *italics* made with asterisks.

This is _italics_ made with underscores.

This is **bold** made with asterisks.

This is __bold__ made with underscores.
```

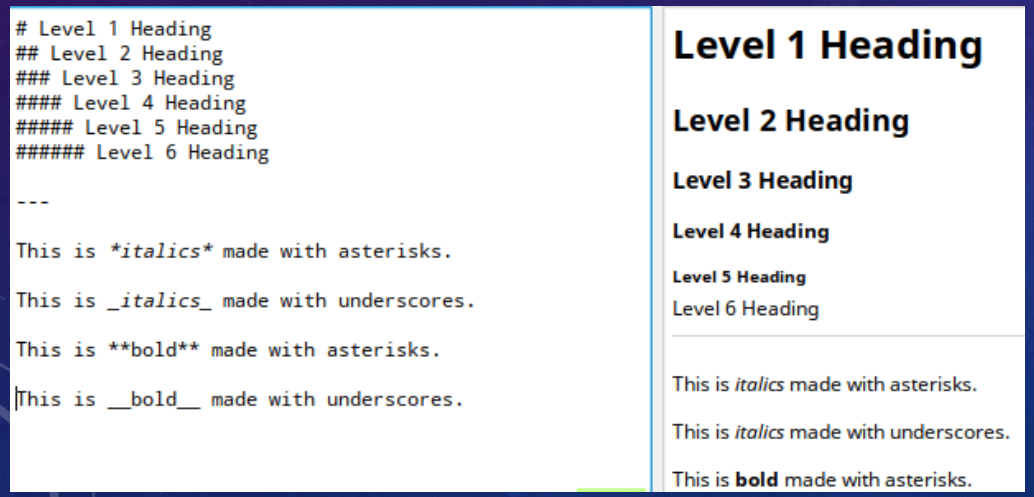

Project documentation will include a **README.md** file in the root directory of your GitHub repository that includes:

- A short description of what is in the repository.
- Instructions how to build and execute the application from the command line, including all environment dependencies.
- Instructions how to execute the unit tests and Playwright tests from the command line.
- A link to your Final Video Presentation.

DOCUMENTATION WRITTEN IN MARKDOWN

Writing Markdown is easy.

I highly recommend writing Markdown using an IDE with a Markdown plugin (all JetBrains IDEs support this), **Visual Studio Code** with a Markdown extension, or a dedicated Markdown editor such as **Typora**.



```
# Level 1 Heading
## Level 2 Heading
### Level 3 Heading
#### Level 4 Heading
##### Level 5 Heading
###### Level 6 Heading

---

This is *italics* made with asterisks.

This is _italics_ made with underscores.

This is **bold** made with asterisks.

This is __bold__ made with underscores.
```

Level 1 Heading

Level 2 Heading

Level 3 Heading

Level 4 Heading

Level 5 Heading

Level 6 Heading

This is *italics* made with asterisks.

This is *italics* made with underscores.

This is **bold** made with asterisks.

I am a big fan of Typora. It is cross platform, visually clean, and has strong spell checking (a curiously unique feature in the Markdown world). I recommend giving it a try. I use it for all my work. It is a licensed tool, but the cost is very low ($15!).

Typora

BE OPINIONATED AND PASSIONATE ABOUT THE TOOLS YOU USE... THEY SAY A LOT ABOUT YOU. I ALWAYS ASK WHAT TOOLS PEOPLE LOVE IN INTERVIEWS.

LEARN TO WRITE GREAT DOCUMENTATION

Great engineers are great communicators.

We must reliably communicate very complex ideas to many people.

Learn to write effectively and get the best writing tools you can afford.

You have two important documentation efforts to complete:

- Your project's README.md file, and
- Your final video presentation.

- You get extra credit for visiting the **KSU Writing Center** up to two times. This will help you learn to be a better writer and communicator.
- Towards the end of the semester after you have pounded out your project documentation, go to the KSU Writing Center and have them help you polish your work.

FINAL VIDEO PRESENTATION

Your completed project will include a 5-to-8-minute video presentation.

Your video will demonstrate:

- starting your application from the command line and using it from a web browser, and
- executing your unit tests and playwright tests from a terminal/command line, and
- showing 100% code coverage from your IDE.

Put some effort into this to make it organized, succinct, and interesting.

Consider writing a script and perhaps use a PowerPoint presentation to frame each section of your video. This is another area where the KSU Writing Center can help you.

Every semester someone does a video that just blows me away.

This is a good skill to develop; it helps you showcase your work and promote yourself.

PLEASE AVOID PRODUCING A TERRIBLE, DULL, HARD TO UNDERSTAND VIDEO.
I MUST WATCH DOZENS OF THESE, SO HAVE PITY AND KEEP ME INTERESTED.

SUBMISSION SCHEDULE

The project is due on Friday April 26th at 11:59 PM.

At that time, all GitHub repositories will be automatically cloned by the instructor for grading.

To help you stay on schedule, you will submit *two progress reports* explaining where you are in the project.

- **Progress Report 1 - Monday 3/18**
  I recommend having your web application fully operational by this point.
- **Progress Report 2 - Monday 4/15**
  I recommend having 100% unit test coverage by this point.

Each progress report is 1.5% of your overall project score. You will submit them via a D2L drop box. These are obviously very easy points towards your project.

YOU DO NOT HAVE TO BE ON THE SCHEDULE I RECOMMEND, BUT YOU SHOULD DO YOUR BEST TO BE CLOSE. WAITING UNTIL THE LAST MINUTE TO DO THIS PROJECT WILL LIKELY RESULT IN A VERY POOR GRADE.

YOUR FINISHED PROJECT WILL INCLUDE

- A URL to your public team GitHub repository.
- Working source code for your Calculator web application checked into your public team GitHub repository. Your source code will include *all assets necessary* to compile and execute your project.
- Working unit tests providing 100% coverage of all Calculator logic (some methods require multiple tests to test all requirements, such as division to check both the division function *and* detecting division by zero).
- Working end-to-end Playwright tests testing various Calculator user interface functions including several single and double operand functions, the clear function, "not a number" conditions, and invalid input.
- A detailed README.md written in Markdown that explains the environment configuration and steps to execute your Calculator web application and tests from the command line. This file will also include a link to your final video presentation.
- A video presentation that demonstrates your completed application, unit tests, and 100% coverage in your IDE.

EVERY SEMESTER SOMEONE SUBMITS INCOMPLETE SOURCE CODE OR SOURCE THAT IS FULL OF COMPILER ERRORS. DO NOT BE THAT PERSON. WORKING CODE IS WHAT SOFTWARE ENGINEERS PRODUCE.

CODE THAT WILL NOT COMPILE WILL AUTOMATICALLY REDUCE YOUR PROJECT SCORE BY 25% OR MORE (DEPENDING ON SEVERITY).

GRADING

This project is 25% of your semester grade, so it is essential that you give yourself enough time to complete it.

Your final grade can only be one letter grade higher than your project. If you do poorly on the project, you will do poorly overall.

If your project meets the requirements in this document, not only will you receive a top score, but you will have earned valuable expertise and enhanced your GitHub repository with a great looking project.

YOUR COMPLETED PROJECT IS DUE FRIDAY 4/26 AT 11:59 PM

COLLABORATION

You can collaborate with your teammate, but not with other teams. Each team's work must be their own.

If you choose to work alone, you are committing to that decision for the duration of the project. You cannot change your mind later.

When you are stuck, you are welcome to ask for advice from classmates or visit the KSU computer lab to get you moving again, *but your work must be your own.*

Do not consider outsourcing your project to a homework help site like Chegg. I have had that happen before. I am pretty good at spotting unusual work and I will be reading your code.

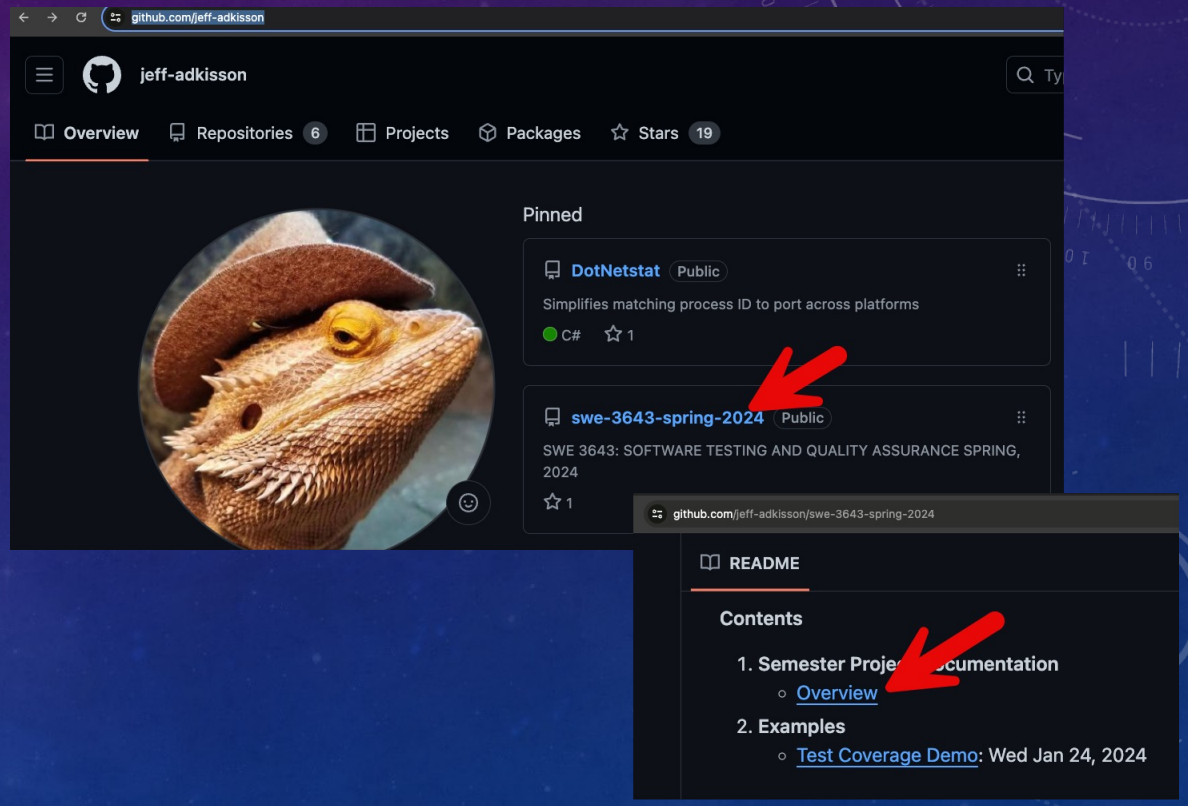
AI tools such as ChatGPT cannot help you much with this project. But even if they can, you are cheating yourself of valuable experience you will need after you graduate.

DO YOUR OWN WORK TO LEARN HOW TO BE A TOP ENGINEER.

PROJECT DOCUMENTATION

All documentation for this project is available in Jeff's 3643 GitHub repository.

- I recommend bookmarking the documentation and reading it very carefully.
- Pay particular attention to the submission guidelines, submission dates, grading rubrics.



https://github.com/jeff-adkisson/swe-3643-spring-2024/blob/master/project/README.md

WHEN YOU NEED HELP WITH CODE

- **I will try to assist you, but I will not write your code.**
- Come to some or all the project workdays at the end of the semester for help. See the Syllabus for the dates. Last minute frantic emails just before the project is due are unlikely to get a helpful response.
- When you are stuck, you are welcome to ask for advice from classmates or visit the KSU computer lab to get you moving again, *but your work must be your own.*
- I am not a miracle worker. You need to do your best to solve your problem (or at least understand it) before asking for help.
- I am most proficient with C#, so consider picking that language if you anticipate needing a lot of help.

When you ask me for coding assistance:

- Be very specific. If you are vague, I cannot help you. ”It’s not working.” is not specific.
- Always include your team’s GitHub repository URL.
- Check the offending code into your GitHub repository so I can look at it and possibly run it. If your code is not in your GitHub repository, I will not attempt to compile or run it. Also consider creating a separate source branch (do some reading if that means nothing) before sharing problematic code with me to avoid breaking operational code.
- To compile and run your code, your **README.md** must have the environment configuration instructions and execution steps. Keep those accurate and up-to-date.

GET STARTED IMMEDIATELY

1. Pick your team and submit your team members and GitHub repository URL to the project spreadsheet by Thursday, February 8th.

   https://shorturl.at/eAOSW

   ** This is your first project grade. Do not miss this date.*

2. Read the **Project Documentation** thoroughly.

   https://github.com/jeff-adkisson/swe-3643-spring-2024/blob/master/project/README.md

3. Do some research, then pick a language and web server framework from the available choices.

   https://github.com/jeff-adkisson/swe-3643-spring-2024/blob/master/project/README.md#languages-web-server-architectures-and-test-runners

4. Start coding. Ask questions at every lecture.
5. Submit your Progress Reports via D2L.
   - Progress Report 1 - Monday 3/18

     I recommend having your web application fully operational by this point.
   - Progress Report 2 - Monday 4/15

     I recommend having 100% unit test coverage by this point.
6. Come to some or all the project workdays at the end of the semester (see Syllabus for dates) if you need help.

   If you show up with nothing done on these dates, there is not much I can do to help you.
7. Check-in your completed project to your team GitHub repo by Friday 4/26 at 11:59 PM. All repos will be cloned by the instructor at midnight.